# Practical Machine Learning HAR Project

a student

July 7, 2016

## Overview and summary

This project applies bagging approach to the provided training data to predict the quality of a weight-lifting exercise based on measurements from three sensors worn by human subjects, plus one sensor placed on dumbbells.

The analytic approach was guided in part by the descriptions provided in the paper by Velloso et al, "Qualitative Activity Recognition of Weight Lifting Exercises", referenced in the Coursera assignment. The article was helpful in selecting features from among the 160 variables in the dataset. The authors used a random forest model with bagging; I used a random forest with Box-Cox preprocessing and 6-fold cross validation.

The analysis relies on the caret package, as well as the recommendations included in lgreski's Discussion posting "Improving Performance of Random Forest in *caret::train()*", in particular by using parallel processing.

## The Process

I began by invoking the packages to be used in preparing this report: caret, ggplot2, GGally, parallel, doParallel, party, and randomForest. I also set a random number seed for reproducibility.

I then read in the training and testing data tables.

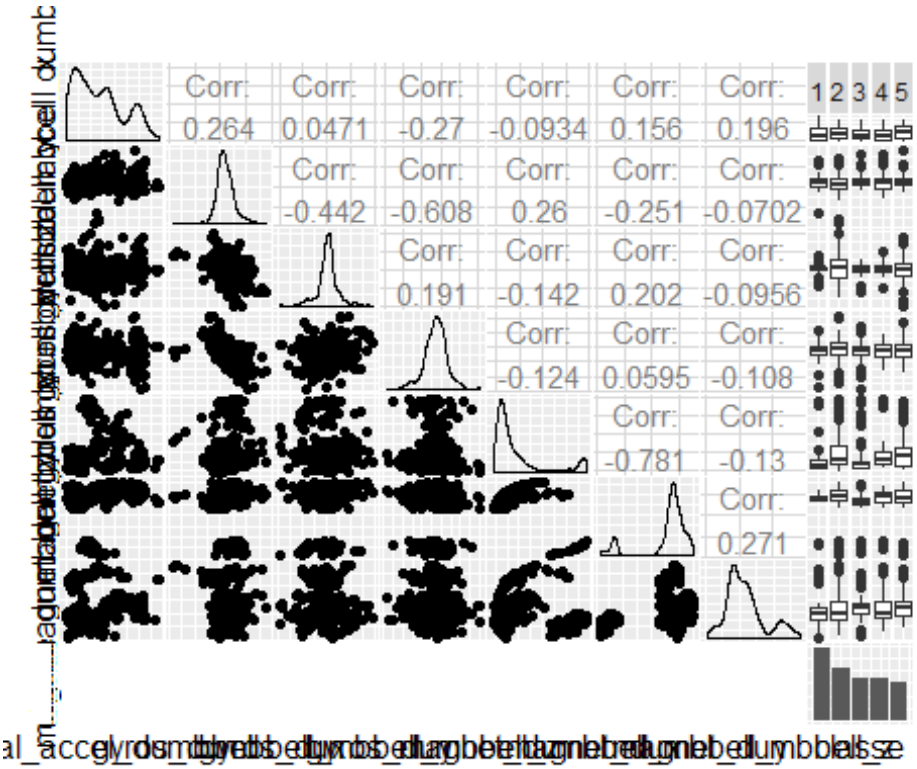
```
##
##     A     B     C     D     E
## 0.284 0.194 0.174 0.164 0.184
```

We see that approximately 28% of the observed lifts were done correctly (Class A), with the remaining cases involved an error of one type or another. Between 16 and 19% fell into each of the four "error" categories.

Guided by the above-referenced article, I explored the available variables selectively for each sensor location, using a 1000 observation simple random subset of the training data for processing speed. For example, the authors reported using 4 measures from the dumbbell taken from the accelerometer, gyroscope and magnenometer. I selected 7 of the dumbbell readings for exploration (see figure below), and based on that inspection chose two of them ( magnet_dumbbell_y and magnet_dumbbell_z) for inclusion as features.

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




After similar explorations with the other three placements, I chose 11 features for the random forest model taken variously from the belt, forearm, arm and dumbbells. I then create a y vector containing the classe observations from the training set, and an x object with all 11 of the selected features.

```
# set up list of features as x
y <- training$classe
# choose x columns after exploratory select columns per article
xbelt <- c(11, 37, 39, 41, 44)
xarm <- c(49, 66)
xdumb <- c(120,121)
xfore <- c(123, 151)
allx <- c(xbelt, xarm, xdumb, xfore)
x <- training[, allx]
```

## Configure parallel processing

Following Greski's advice on a parallel implementation of Random Forest, I configured parallel processing.

```
# configure Parallel Processing
cluster <- makeCluster(detectCores()-1) # convention to leave 1 core for OS
registerDoParallel(cluster)
```

## Configure trainControl Object

The next step is to set parameters for the trainControl object to be referenced in fitting the random forest model. Note that we specify a 6-fold cross validation (method = "cv" and number = 6). Initially I used 10-fold cv, but the even with parallel processing the time required was quite long, so I reduced it to 6 folds.

```
fitControl <- trainControl(method="cv",
                 number = 6,
                 allowParallel = TRUE)
```

## Develop the training model

Here we actually estimate the random forest model. Based on inspection of the features and finding several of them to be skewed, I elected to preprocess with a Box-Cox transformation.

```
fitrf <- train(x,y, data=training,
               preProcess=c("BoxCox"), method="rf",
               trControl = fitControl)
```

## De-register parallel processing cluster

we now call the stopcluster() function, having done the heavy lifting of fitting the model.

```
stopCluster(cluster)
```

## Summary of model performance

At this point we can assess how accurately the training model performs with the training data. The next code chunk summarizes the model and its fit.

```
fitrf

## Random Forest
##
## 19622 samples
##    11 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing: Box-Cox transformation (2)
## Resampling: Cross-Validated (6 fold)
```

```
## Summary of sample sizes: 16351, 16352, 16351, 16352, 16352, 16352, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9589747  0.9480798
##    6    0.9581085  0.9469797
##   11    0.9515342  0.9386609
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

```
fitrf$resample
```

```
##    Accuracy     Kappa Resample
## 1 0.9596330 0.9489127    Fold2
## 2 0.9602568 0.9496763    Fold1
## 3 0.9556710 0.9438983    Fold3
## 4 0.9605505 0.9500893    Fold6
## 5 0.9590214 0.9481450    Fold5
## 6 0.9587156 0.9477574    Fold4
```

```
confusionMatrix.train(fitrf)
```

```
## Cross-Validated (6 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    A    B    C    D    E
##          A 27.9  0.6  0.2  0.2  0.1
##          B  0.1 17.8  0.2  0.1  0.2
##          C  0.2  0.5 16.9  0.7  0.1
##          D  0.2  0.2  0.1 15.4  0.1
##          E  0.0  0.1  0.0  0.0 17.9
##
##  Accuracy (average) : 0.959
```

From the model summary, we see that (perhaps due to overfitting) the accuracy rate is quite high at approximately 95%. It is unrealistic to expect such high accuracy with the testing data, but we now can go ahead and make the estimates required for the second quiz.

## Predictions with Testing Data

```
xnew <- testing[, allx]
newpred <- rffits <- predict(fitrf, xnew)
newpred
```

```
##  [1] B A B A A E D B A A B C B A E B A B B B
## Levels: A B C D E
```